A reduction is a procedure to convert one problem to another problem, in such a way that a solution to the second problem can be used to solve the first problem. The conversion itself must be doable by a Turing Machine that halts. That is, there must be a concrete algorithm for a conversion.

**REDUCTION:**

**problem X reduces to Problem Y**

means

*solving X would be easy, if we knew how to solve Y*

**Example 1.** The problem of having a Ferrari reduces to the problem of obtaining $200K.

**Proof:** Say that we could obtain $200K. The following algorithm leads us to drive a Ferrari:

1. Go to a Ferrari dealer.
2. Buy a Ferrari (trivial, as several good models cost < $200K)

**Corollary.** $200K is hard to obtain.

**Proof.** Because if it was easy, then having a Ferrari would be easy. Since that is not the case, the corollary follows.

**Example 2.** The problem of traveling from Boston to anywhere warm reduces to the problem of deciding on a warm location and buying a plane ticket to that location. This problem of buying a plane ticket reduces to the problem of earning the money for the ticket. This problem of earning the money reduces to the problem of finding a job.

This idea of reducibility plays a part in classifying problems: **if A is reducible to B and B is decidable, A also is decidable.**
**If A is undecidable and reducible to B, B is undecidable.** Why? Because if B is actually decidable then we could build a machine and decide A – therefore B must not be decidable.

\[
\text{HALT}_{TM} = \{ (M, w) \mid M \text{ is a TM and halts on input } w \}
\]

Note: related to the \( A_{TM} \) problem - \( A_{TM} = \{ <M, w> \mid M \text{ is a TM that accepts } w \} \) which determines whether a machine accepts a particular string (acceptance problem)

**Example.** \( \text{HALT}_{TM} = \{ (M, w) \mid M \text{ is a TM and halts on input } w \} \)

**Theorem.** \( \text{HALT}_{TM} \) is undecidable.

**Proof.** We will show that “solving \( A_{TM} \) would be easy, if we could solve \( \text{HALT}_{TM} \).” Then, since we know that solving \( A_{TM} \) is hard, we conclude that solving \( \text{HALT}_{TM} \) is hard.

More precisely, we will **reduce** \( A_{TM} \) to \( \text{HALT}_{TM} \), which means that if \( \text{HALT}_{TM} \) is decidable, then \( A_{TM} \) is decidable—which it isn’t.

![Diagram](image)

Let a machine \( H \) decide \( \text{HALT}_{TM} \). Here is a machine \( S \) that decides \( A_{TM} \):

\[
A := \text{“On input } (M, w)\text{,}
\begin{align*}
1 & \text{ Run } H \text{ on input } (M, w) \\
2 & \text{ If } H \text{ rejects, reject} \\
3 & \text{ If } H \text{ accepts, simulate } M \text{ on input } w \text{ until it halts} \\
4 & \text{ If } M \text{ accepts, accept; if } M \text{ rejects, reject”}
\end{align*}
\]

Proof by contradiction. How?

**Classifying the Computability of Problems**

A language can be in one of the following categories:

- **Decidable**
  These are the problems that we call **solvable by an algorithm**. To prove a language decidable, find an algorithm run by a Turing Machine that always halts.
• **Recognizable**
  To prove a language recognizable, find a Turing Machine that accepts all strings of the language and does not accept any strings out of the language. The Turing Machine is allowed to loop forever instead of rejecting.

  To prove the language not decidable, use a reduction from a recognizable problem that is not decidable.

• **Non-recognizable**
  Those are the hopeless problems! To prove a language is not recognizable, show that it is the complement of a language that is not decidable. Or, use a reduction from a non-recognizable language.

\[ E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \]

**Theorem.** \( E_{TM} \) is undecidable.

**Proof.** We will reduce \( A_{TM} \) to \( E_{TM} \).
Say that there is a machine \( R \) deciding \( E_{TM} \).
So \( R \) can decide, given a machine description \( (M) \), whether \( L(M) = \emptyset \).

For a given string \( w \), consider the following machine:

\[ M'(w) := \text{“On input x,} \]
1. If \( x \neq w \), reject.
2. Otherwise, simulate \( M \) on \( w \).
   - If \( M \) accepts, *accept*.
   - If \( M \) rejects, *reject*."

Notice that \( M'(w) \) has \( w \) hardwired in it. What is \( L(M'(w)) \)?
It is simply \( \{ w \} \), if \( M \) accepts \( w \).
But it is \( \emptyset \) if \( M \) does not accept \( w \).

Given the description of \( M' \) as an input, \( R \) accepts if \( M \) accepts \( w \), and rejects if \( M \) rejects \( w \). So here is a machine that decides \( A_{TM} \):

\[ S := \text{“On input } (M, w), \]

\[ \text{construct machine } M'(w) \]
\[ \text{machine } S \text{ to decide } A_{TM} \]

\[ \begin{array}{c}
\text{reject accept} \\
\text{accept reject}
\end{array} \]
1. Construct the machine $M'(w)$ as described above
2. Run $R$ ($E_{TM}$) on input $<M'>$
3. If $R$ accepts, reject. If $R$ rejects, accept.”

reject – means its empty or $w \neq x$
accept – means its not empty and $x = w$

Since $R$ always halts and decides whether $L(M'(w)) = \emptyset$, $S$ always halts and decides whether $M$ accepts $w$.

$EQ_{TM} = \{ M, N | M$ and $N$ are TMs, and $L(M) = L(N) \}$

Theorem. $EQ_{TM}$ is undecidable.

Proof. We will reduce $E_{TM}$ to $EQ_{TM}$.

Say that a machine $R$ decides $EQ_{TM}$. Here is a machine $S$ that decides $E_{TM}$:

$S := \text{“On input } M, \text{ where } M \text{ is a TM,}
1. \text{Run } R \text{ on input } (M, M_1) \text{ where } M_1 \text{ is a trivial TM that rejects all inputs.}
2. \text{If } R \text{ accepts, accept. If } R \text{ rejects, reject.”}$

Proof by contradiction: if $EQ_{TM}$ is possible then it would be possible (by simple reduction) to create $E_{TM}$ - however we know that $E_{TM}$ is undecidable therefore it must be impossible to create $EQ_{TM}$

Theorem. The complement of $EQ_{TM}$, $\overline{EQ_{TM}} = \{ M, N | M$ and $N$ are TMs and $L(M) \neq L(N) \}$ is undecidable.

Proof. Say it is decidable, and $R$ is a machine that on input $(M, N)$, it accepts if $L(M) = L(N)$. We show how to construct $S$ deciding $A_{TM}$:

$S := \text{“On input } (M, w), \text{ Construct the descriptions of the following machines:}$$M_1 := \text{“On any input, reject”}$
M_2 := “On any input,
    Simulate M on w, and if M accepts, accept.”
Pass (M_1, M_2) to R.
If R accepts, accept. If it rejects, reject.”

If R accepts, then L(M_2) = \{w\}, so M accepts w.

Corollary. Therefore, both EQ_{TM} and \overline{EQ_{TM}} are non-recognizable.

**Mapping Reducibility**

Being able to reduce problem A to problem B by using a mapping reducibility means that a computable function (reduction) exists that converts instances of problem A to instances of problem B.

**Computable functions**

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a computable function if some Turing machine M, on every input w, halts with just f(w) on its tape.

- All math operations are computable functions. It is possible to make a machine that takes <M,N> and halts with M+N on its tape.
- Computable functions may be transformations of machine descriptions.

**Formal Definition of Mapping Reducibility**

Language A is mapping reducible to language B (\( A \leq_m B \)) if there is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \), where for every \( w \),

\[
    w \in A \iff f(w) \in B
\]

The function f is called the reduction of A to B.
Theorem 5.22
If \( A \leq_m B \) and \( B \) is decidable, then \( A \) is decidable.

Corollary 5.23
If \( A \leq_m B \) and \( A \) is undecidable, then \( B \) is undecidable.

Theorem 5.28
If \( A \leq_m B \) and \( B \) is Turing-recognizable, then \( A \) is Turing-recognizable.

Corollary 5.29
If \( A \leq_m B \) and \( A \) is not Turing-recognizable, then \( B \) is not Turing-recognizable.

Example 5.24 Reduction from \( A_{TM} \) to \( HALT_{TM} \) to prove \( HALT_{TM} \) is undecidable

Previously used:
Mapping reducibility from $A_{TM}$ to $HALT_{TM}$ — must present a computable function $f$ that takes $<M,w>$ and returns output $<M',w'>$, where

$\langle M, w \rangle \in A_{TM}$ if and only if $\langle M', w' \rangle \in HALT_{TM}$

$f = \text{“On input } <M,w>:\n 1. \text{Construct the following machine } M'\n     M' = \text{“On input } x:\n      1. \text{Run } M \text{ on } x.\n      2. \text{If } M \text{ accepts, accept.}\n      3. \text{If } M \text{ rejects, enter a loop.”}\n 2. \text{Output } <M',w'>.$